Diverse and Additive Cartesian Abstraction Heuristics

Jendrik Seipp and **Malte Helmert**
Universität Basel
Basel, Switzerland
{jendrik.seipp,malte.helmert}@unibas.ch

Abstract

We have recently shown how counterexample-guided abstraction refinement can be used to derive informative Cartesian abstraction heuristics for optimal classical planning. In this work we introduce two methods for producing *diverse* sets of heuristics within this framework, one based on goal facts, the other based on landmarks. In order to sum the heuristic estimates *admissibly* we present a novel way of finding *cost partitionings* for explicitly represented abstraction heuristics. We show that the resulting heuristics outperform other state-of-the-art abstraction heuristics on many benchmark domains.

Introduction

Recently, we presented an algorithm (Seipp and Helmert 2013) for deriving admissible heuristics for classical planning based on the counterexample-guided abstraction refinement (CEGAR) methodology (Clarke et al. 2000). Starting from a coarse abstraction of a planning task, the algorithm iteratively computes an optimal abstract solution, checks if and why it fails for the concrete planning task and refines it so that the same failure cannot occur in future iterations. After a given time or memory limit is hit, the resulting Cartesian abstraction is used as an admissible search heuristic.

As the number of CEGAR iterations grows, one can observe diminishing returns: it takes more and more iterations to obtain further improvements in heuristic value. Therefore, in this work we propose building multiple smaller *additive* abstractions instead of a single big one.

The standard way of composing admissible heuristics is to use the maximum of their estimates. This combination is always admissible if the component heuristics are. In order to gain a more informed heuristic it would almost always be preferable to use the *sum* of the estimates, but this estimate is often not admissible. To remedy this problem, we can use *cost partitioning* to ensure that each operator's cost is distributed among the heuristics in a way that makes the sum of their estimates admissible.

The notion of cost partitioning has been formally introduced by Katz and Domshlak (2008). They formulate linear programs that find an optimal admissible cost partitioning for a given search state and any number of abstraction heuristics in polynomial time. The practical use of this result is limited, however, since evaluating the linear programs is computationally very expensive (Pommerening, Röger, and Helmert 2013).

Yang et al. (2008) provide an overview of the work on additive abstractions and some formal proofs concerning cost partitionings. They show that distributing the cost among the heuristics reduces the solving time for many combinatorial puzzles, but do not provide an algorithm for finding cost partitionings.

In this work we introduce the *saturated cost partitioning* algorithm that computes cost partitionings for explicitly represented abstraction heuristics. Iteratively, we find an abstraction, reduce the operator costs so that the heuristic estimates do not change and use the remaining costs for the next abstraction. While our algorithm increases the number of solved benchmark tasks compared to using a single Cartesian abstraction, the resulting abstractions focus on mostly the same parts of the task. Therefore, we propose two methods for producing more *diverse* sets of abstractions. The first strategy computes abstractions for all goal facts separately, while the second does so for all causal fact landmarks of the delete-relaxation of the task (Keyder, Richter, and Helmert 2010).

We show that the construction of multiple abstractions in general and the use of landmarks to diversify the heuristics in particular both lead to a significantly higher number of solved tasks and let heuristics based on Cartesian abstractions outperform many other state-of-the-art abstraction heuristics.

The remainder of the paper is organized as follows: first we give some formal definitions and present our algorithm for finding cost partitionings for abstraction heuristics. Then we show that combining multiple heuristics found with the plain CEGAR algorithm does not improve the overall performance much since the calculated abstractions are too similar. Therefore, we introduce several methods for finding more diverse abstractions that concentrate on different aspects of the task. Afterwards, we evaluate our ideas experimentally and conclude.

Background

We consider optimal planning in the classical setting, using a SAS$^+$-like (Bäckström and Nebel 1995) representation.

Definition 1. ***Planning tasks.***
*A **planning task** is a 5-tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, c, s_0, s_\star \rangle$, where:*

- *$\mathcal{V}$ is a finite set of **state variables** v, each with an associated finite domain $\mathcal{D}(v)$.*
  *A **fact** is a pair $\langle v, d \rangle$ with $v \in \mathcal{V}$ and $d \in \mathcal{D}(v)$.*
  *A **partial state** is a function s defined on a subset of $\mathcal{V}$. This subset is denoted by $\mathcal{V}_s$. For all $v \in \mathcal{V}_s$, we must have $s(v) \in \mathcal{D}(v)$. Partial states defined on all variables are called **states**, and $\mathcal{S}(\Pi)$ is the set of all states of Π. Where notationally convenient we treat states as sets of facts.*
  *The **update** of partial state s with partial state t, $s \oplus t$, is the partial state defined on $\mathcal{V}_s \cup \mathcal{V}_t$ which agrees with t on all $v \in \mathcal{V}_t$ and with s on all $v \in \mathcal{V}_s \setminus \mathcal{V}_t$.*
- *$\mathcal{O}$ is a finite set of **operators**. Each operator o has a **precondition** $pre(o)$ and **effect** $eff(o)$, which are partial states. The **cost function** c assigns a cost $c(o) \in \mathbb{N}_0$ to each operator.*
- *$s_0 \in \mathcal{S}(\Pi)$ is the **initial state** and $s_\star$ is a partial state, the **goal**.*

The notion of transition systems is central for assigning semantics to planning tasks:

Definition 2. ***Transition systems and plans.***
*A **transition system** $\mathcal{T} = \langle S, T, s_0, S_\star \rangle$ consists of a finite set of **states** S, a set of **transitions** T, an **initial state** $s_0 \in S$ and a set of **goal states** $S_\star \subseteq S$. A transition $s \xrightarrow{l,w} s' \in T$ from state $s \in S$ to state $s' \in S$ has an associated **label** l and non-negative **weight** w.*

*A path from $s \in S$ to any $s_\star \in S_\star$ following the transitions is a **plan** for s. A plan is **optimal** if the sum of weights along the path is minimal.*

A planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, c, s_0, s_\star \rangle$ induces a transition system with states $\mathcal{S}(\Pi)$, initial state s_0, goal states $\{s \in \mathcal{S}(\Pi) \mid s_\star \subseteq s\}$ and transitions $\{s \xrightarrow{o,c(o)} s \oplus eff(o) \mid s \in \mathcal{S}(\Pi), o \in \mathcal{O}, pre(o) \subseteq s\}$. Optimal planning is the problem of finding an optimal plan in the transition system induced by a planning task starting in s_0, or proving that no such plan exists.

By losing some distinctions between states we can create an *abstraction* of a planning task. This allows us to obtain a more "coarse-grained", and hence smaller, transition system.

Definition 3. ***Abstraction.***
*An **abstraction** of a transition system $\mathcal{T} = \langle S, T, s_0, S_\star \rangle$ is a pair $\mathcal{A} = \langle \mathcal{T}', \alpha \rangle$ where $\mathcal{T}' = \langle S', T', s'_0, S'_\star \rangle$ is a transition system called the **abstract transition system** and $\alpha : S \to S'$ is a function called the **abstraction mapping**, such that $\alpha(s) \xrightarrow{l,w} \alpha(s') \in T'$ for all $s \xrightarrow{l,w} s' \in T$, $\alpha(s_0) = s'_0$, and $\alpha(s_\star) \in S'_\star$ for all $s_\star \in S_\star$.*

Abstraction preserves paths in the transition system and can therefore be used to define admissible and consistent heuristics for planning. Specifically, $h^{\mathcal{A}}(s)$, the heuristic estimate for a concrete state $s \in \mathcal{S}$, is defined as the cost of an optimal plan starting from $\alpha(s) \in S$ in the abstract transition system.

Saturated Cost Partitioning

Using only a single abstraction of a given task is often not enough to cover all or at least most important parts of the task in reasonable time. Therefore, it is often beneficial to build multiple abstractions that focus on different aspects of the problem (Holte et al. 2006). Since we want the resulting heuristics to be additive in order to obtain a more informed overall estimate, we have to ensure that the sum of their individual estimates is admissible. One way of doing so is to use a *cost partitioning* that divides operator costs among multiple cost functions:

Definition 4. ***Cost partitioning.***
*A **cost partitioning** for a planning task with operator set $\mathcal{O}$ and cost function c is a sequence $c_1, \ldots, c_n$ of cost functions $c_i : \mathcal{O} \to \mathbb{N}_0$ that assign costs to operators $o \in \mathcal{O}$ such that $\sum_{1 \leq i \leq n} c_i(o) \leq c(o)$ for all $o \in \mathcal{O}$.*

Cost partitioning can be used to enforce additivity of a group of heuristics $h_1, \ldots, h_n$. Each heuristic h_i is evaluated on a copy of the planning task with operator cost function c_i. If each h_i is admissible for this planning task, then their sum $\sum_{i=1}^{n} h_i$ is admissible for the original planning task due to the way the operator costs are "split" by the cost partitioning.

The question is: how do we find a cost partitioning that achieves a high overall heuristic estimate? Our *saturated cost partitioning* algorithm iteratively computes h_i and associates with it the minimum cost function c_i that preserves all of h_i's estimates. Therefore, each cost function c_i only uses the costs that are actually needed to prove the estimates made by h_i, and the remaining costs are used to define further cost functions and heuristics that can be admissibly added to h_i.

The following observation forms the basis of our algorithm: given an abstract transition system, we can often reduce transition weights without changing any goal distances. An example of this situation is shown in Figure 1. If we ignore the numbers in brackets and operator labels for now, we can see that for example the transition from $h = 2$ to $h = 0$ with original weight 5 can be assigned a weight of 2 without affecting any goal distances. We formalize the general insight in the following lemma:

Lemma 5. ***Distance-preserving weight reduction.***
Consider transition systems $\mathcal{T}$ and $\mathcal{T}'$ that only differ in the weight of a single transition $a \to b$, which is w in $\mathcal{T}$ and w' in $\mathcal{T}'$. Let h and h' denote the goal distance functions in $\mathcal{T}$ and $\mathcal{T}'$.
If $h(a) - h(b) \leq w' \leq w$, then $h = h'$.

Proof. $\mathcal{T}$ and $\mathcal{T}'$ only differ in the weight of $a \to b$, so it suffices to show $h'(a) = h(a)$. We have $h'(a) \leq h(a)$ because $w' \leq w$. It remains to show $h'(a) \geq h(a)$.

Clearly $h'(b) = h(b)$: we can assume that shortest paths from b are acyclic, hence do not use the transition $a \to b$, and all other transitions have the same cost in $\mathcal{T}'$ and $\mathcal{T}$.

If a shortest path from a in $\mathcal{T}'$ does not use $a \to b$, then clearly $h'(a) = h(a)$. If it does, then its cost is $h'(a) = w' + h'(b) = w' + h(b) \geq h(a) - h(b) + h(b) = h(a)$. □

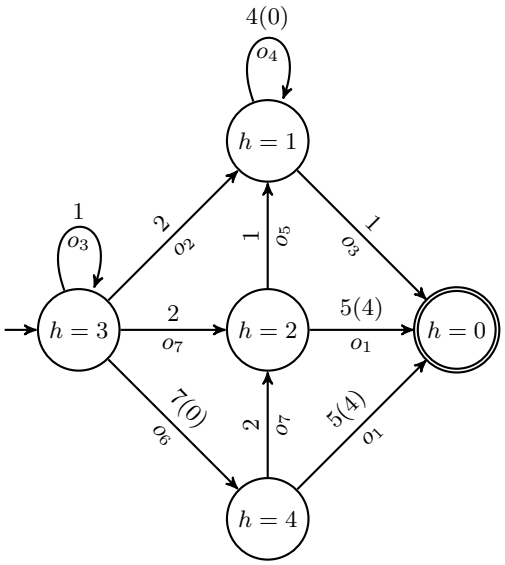

Figure 1: Abstract transition system of an example planning task. Every transition is associated with an operator and a weight corresponding to the operator's cost. The numbers in brackets show the reduced operator costs that suffice to preserve all goal distances.

In transition systems of planning task abstractions, weights are induced by operator costs. Lemma 5 therefore implies that we can reduce the cost of some operators without changing the heuristic estimates of the abstraction. One option is to substitute the task's cost function with the *saturated cost function*:

Definition 6. ***Saturated cost function.*** *Let Π be a planning task with operator set $\mathcal{O}$ and cost function c. Furthermore, let $\mathcal{T}$ be an abstract transition system for Π with transitions T and goal distance function h.*

Then we define the saturated cost function $\hat{c}(o)$ *for $o \in \mathcal{O}$ as $\hat{c}(o) = \max_{a \xrightarrow{o,w} b \in T} \max\{0, h(a) - h(b)\}$.*

The saturated cost function assigns to each operator the minimum cost that preserves all abstract goal distances. In the example abstraction in Figure 1 we show the reduced operator costs assigned by the saturated cost function in brackets. Note that the transition from $h = 2$ to $h = 0$ must have at least a weight of 4 when we take the operator labels into account. Otherwise, the transition between $h = 4$ and $h = 0$, which is induced by the same operator, would also be assigned a weight smaller than 4 and thus the goal distance for $h = 4$ would decrease.

Theorem 7. ***Minimum distance-preserving cost function.*** *Let Π be a planning task with operator set $\mathcal{O}$ and cost function c. Furthermore, let $\mathcal{T}$ be an abstract transition system for Π with transitions T and goal distance function h. Then for the saturated cost function $\hat{c}$ we have:*

1. *$\hat{c}$ preserves the goal distances of all abstract states.*
2. *For all other cost functions c' that preserve all goal distances we have $c'(o) \geq \hat{c}(o)$ for all operators $o \in \mathcal{O}$.*

Proof. 1. Starting from the transition system $\mathcal{T}$, we can repeatedly apply Lemma 5 to reduce the weight of every transition $a \to b$ to $\max\{0, h(a) - h(b)\}$ without affecting the h values. Let $\mathcal{T}'$ be the resulting transition system. In a second step, we replace the weights of all transitions $a \xrightarrow{o,w} b$ of $\mathcal{T}'$ by the *maximum* weight of all transitions with label o, which is the saturated cost $\hat{c}(o)$. Let $\mathcal{T}''$ be the resulting transition systems.
By Lemma 5, the goal distances in $\mathcal{T}$ and $\mathcal{T}'$ are the same. All label weights in $\mathcal{T}''$ are bounded by the label weights of $\mathcal{T}'$ from below and $\mathcal{T}$ from above, which means that goal distances in $\mathcal{T}''$ must also be the same.

2. By contradiction: let $c'(o) < \hat{c}(o)$ for some operator $o \in \mathcal{O}$. By definition of $\hat{c}(o)$ and because costs must be non-negative, this means $c'(o) < \max_{a \xrightarrow{o,w} b \in T}(h(a) - h(b))$, and hence there exists a transition $a \xrightarrow{o,w} b \in T$ with $c'(o) < h(a) - h(b)$. This implies $h(a) > c'(o) + h(b)$, which violates the triangle inequality for shortest paths in graphs.

□

Splitting the cost of each operator o into the cost needed for preserving the goal distances $\hat{c}(o)$ and the remaining cost $c(o) - \hat{c}(o)$ produces the desired cost partitioning: after heuristic h has been computed we associate with it the saturated cost function $\hat{c}(o)$ and use the remaining operator cost $c(o) - \hat{c}(o)$ to define further heuristics that can be admissibly added to h.

The procedure can be used for any abstraction heuristic with a suitably small number of abstract states and where the transition system is either explicitly represented or easily enumerable (such as pattern databases). We apply it to Cartesian abstractions (Seipp and Helmert 2013), which use explicitly represented transition systems.

Multiple Abstractions

Having discussed how we can combine multiple abstraction heuristics, the natural question is how to come up with different abstractions to combine. As discussed previously, we build on our earlier CEGAR approach for Cartesian abstractions (Seipp and Helmert 2013). In our earlier work, we used a timeout of 900 seconds to generate the single abstraction. The simplest idea to come up with n additive abstractions, then, is to repeat the CEGAR algorithm n times with timeouts of $900/n$ seconds, computing the saturated cost function after each iteration, and using the remaining cost in subsequent iterations.

Table 1 shows the number of solved tasks from previous IPC challenges for different values of n. All versions are given a time limit of 30 minutes (of which at most 15 minutes are used to construct the abstractions) and 2 GB of memory to find a solution.

We see that increasing the number of abstractions from 1 to 2 is mildly detrimental. It increases coverage in 2 out of 44 domains, but reduces coverage in 6 domains. The total coverage decreases from 562 to 559. However, using even more abstractions increases coverage, with peaks around 566 solved tasks for 10–20 abstractions. It is also

	Abstractions					
Coverage	1	2	5	10	20	50
airport (50)	19	19	20	19	20	**21**
driverlog (20)	10	10	10	**11**	10	10
logistics-00 (28)	14	**16**	**16**	**16**	**16**	**16**
logistics-98 (35)	3	**4**	**4**	**4**	**4**	**4**
miconic (150)	55	55	**56**	**56**	55	55
mprime (35)	**27**	26	26	26	26	25
nomystery-11 (20)	**10**	9	**10**	**10**	**10**	9
pipesworld-t (50)	**11**	**11**	**11**	**11**	**11**	10
rovers (40)	6	6	6	**7**	**7**	**7**
sokoban-08 (30)	**21**	20	20	20	20	19
sokoban-11 (20)	**18**	17	17	17	**18**	16
tidybot-11 (20)	13	13	13	**14**	**14**	**14**
trucks (30)	6	6	**7**	**7**	**7**	**7**
wood-08 (30)	9	9	9	9	9	**10**
wood-11 (20)	**5**	4	4	4	4	4
zenotravel (20)	**9**	8	**9**	**9**	**9**	**9**
...	...	...	...	...	...	...
Sum (1396)	562	559	564	**566**	**566**	562

Table 1: Number of solved tasks for a growing number of Cartesian abstractions. Domains in which coverage does not change are omitted. Best results are highlighted in bold.

	Abstractions					
Coverage	1	2	5	10	20	50
airport (50)	19	19	**20**	**20**	**20**	**20**
driverlog (20)	**10**	**10**	**10**	**10**	9	9
logistics-00 (28)	15	**18**	**18**	**18**	**18**	17
logistics-98 (35)	4	**5**	**5**	**5**	**5**	**5**
miconic (150)	55	58	**59**	**59**	**59**	58
mprime (35)	**26**	**26**	25	25	25	25
nomystery-11 (20)	9	10	11	**12**	**12**	9
pipesworld-nt (50)	**16**	15	15	15	15	15
pipesworld-t (50)	**12**	11	11	11	11	11
rovers (40)	6	**7**	**7**	**7**	**7**	**7**
sokoban-08 (30)	**21**	**21**	20	20	20	20
sokoban-11 (20)	**18**	**18**	17	17	17	16
tidybot-11 (20)	13	13	**14**	**14**	**14**	**14**
tpp (30)	6	**7**	**7**	**7**	**7**	**7**
trucks (30)	6	**9**	**9**	**9**	**9**	**9**
...	...	...	...	...	...	...
Sum (1396)	565	576	577	**578**	577	571

Table 2: Number of solved tasks for a growing number of Cartesian abstractions preferring to refine facts with higher h^{add} values. Domains in which coverage does not change are omitted. Best results are highlighted in bold.

apparent that performance drops off when too many abstractions are used.

Overall, we note that using more Cartesian abstractions can somewhat increase the number of solved tasks, but computing too many abstractions is not beneficial. We hypothesize that this is the case because the computed abstractions are too similar to each other, focusing mostly on the same parts of the problem. Computing more abstractions does not yield a more informed additive heuristic and instead just consumes time that could have been used to produce fewer, but more informed component heuristics.

To see why diversification of abstractions is essential, consider the extreme case where two component heuristics h_1 and h_2 are based on exactly the same abstraction. Let c_1 and c_2 be the corresponding cost functions. Then the sum of heuristics $h_1 + h_2$ is *dominated* by the heuristic that would be obtained by using the same abstraction only once with cost function $c_1 + c_2$. (This follows from the admissibility of cost partitioning.) So we need to make sure that the abstractions computed in different iterations of the algorithm are sufficiently different.

There are several possible ways of ensuring such diversity within the CEGAR framework. One way is to make sure that different iterations of the CEGAR algorithm produce different results even when presented with the same input planning task. This is quite possible to do because the CEGAR algorithm has several choice points that affect its outcome, in particular in the refinement step where there are frequently multiple flaws to choose from. By ensuring that these choices are resolved differently in different iterations of the algorithm, we can achieve some degree of diversification. We call this approach *diversification by refinement strategy*.

Another way of ensuring diversity, even in the case where the CEGAR algorithm always generates the same abstraction when faced with the same input task, is to modify the inputs to the CEGAR algorithm. Rather than feeding the actual planning task to the CEGAR algorithm, we can present it with different "subproblems" in every iteration, so that it will naturally generate different results. To ensure that the resulting heuristic is admissible, it is sufficient that every subproblem we use as an input to the CEGAR algorithm is itself an abstraction of the original task. We call this approach *diversification by task modification*. We will discuss these two approaches in the following sections.

Diversification by Refinement Strategy

A simple idea for diversification by refinement strategy is to let CEGAR prefer refining for facts with a higher h^{add} value (Bonet and Geffner 2001), because this refinement strategy (unlike the strategy used in our original CEGAR algorithm) is affected by the costs of the operators, which change from iteration to iteration as costs are used up by previously computed abstractions. This inherently biases CEGAR towards regions of the state space where operators still have high costs.

Table 2 shows the results for this approach. We see that the h^{add}-based refinement strategy leads to better results than the original CEGAR algorithm on average: 3 more tasks are solved in the basic case of only one abstraction, and for larger values of n we obtain 9–17 additional solved tasks compared to the corresponding columns in Table 1. We also see that the best values of n lead to a larger improvement over a single abstraction (+13 tasks) than with the original refinement strategy (+4 tasks). However, as in Table 1, the

maximum number of solved tasks is obtained with 10–20 abstractions and calculating more than that leads to a decrease in total coverage.

Overall, we see that using a refinement strategy that takes into account the operator costs and hence interacts well with cost partitioning can lead to better scalability for additive CEGAR heuristics. However, the improvements obtained in this way are quite modest, which motivates the alternative approach for diversification which we discuss next.

Diversification by Task Modification

Diversification by task modification is a somewhat more drastic approach than diversification by refinement strategy. The basic idea is that we identify different aspects of the planning task and then generate an abstraction of the original task for each of these aspects. Each invocation of the CEGAR algorithm uses one of these abstractions as its input and is thus constrained to exclusively focus on one aspect.

We propose two different ways for coming up with such "focused subproblems": *abstraction by goals* and *abstraction by landmarks*.

Abstraction by Goals

Our first approach, abstraction by goals, generates one abstract task for each goal fact of the planning task. The number of abstractions generated is hence equal to the number of goals.

If $\langle v, d\rangle$ is a goal fact, we create a modified planning task which is identical to the original one except that $\langle v, d\rangle$ is the only goal fact. This means that the original and modified task have exactly the same states and transitions and only differ in their goal states: in the original task, *all* goals need to be satisfied in a goal state, but in the modified one, *only* $\langle v, d\rangle$ needs to be reached. The goal states of the modified task are hence a superset of the original goal states, and we can conclude that the modification defines an abstraction in the sense of Def. 3 (where the abstraction mapping α is the identity function).

Abstracting by goals has the obvious drawback that it only works for tasks with more than one goal fact. Since any task could potentially be reformulated to only contain a single goal fact, a smarter way of diversification is desirable.

Abstraction by Landmarks

Our next diversification strategy solves this problem by using *fact landmarks* instead of goal facts to define subproblems of a task. Fact landmarks are facts that have to be true at least once in all plans for a given task (e. g., Hoffmann, Porteous, and Sebastia 2004). Since obviously all goal facts are also landmarks, this method can be seen as a generalization of the previous strategy.

More specifically, we generate the causal fact landmarks of the delete relaxation of the planning task with the algorithm by Keyder, Richter, and Helmert (2010) for finding h^m landmarks with $m = 1$. Then for each landmark $l = \langle v, d\rangle$ we compute a modified task that considers l as the only goal fact.

Without further modifications, however, this change does not constitute an abstraction, and hence the resulting heuristic could be inadmissible. This is because landmarks do not have the same semantics as goals: goals need to be satisfied *at the end* of a plan, but landmarks are only required *at some point* during the execution of a plan.

Existing landmark-based heuristics address this difficulty by remembering which landmarks might have been achieved en route to any given state and only base the heuristic information on landmarks which have not yet been achieved (e. g., Richter, Helmert, and Westphal 2008; Karpas and Domshlak 2009). This makes these heuristics *path-dependent*: their heuristic values are no longer a function of the state alone.

Path-dependency comes at a significant memory cost for storing landmark information, so we propose an alternative approach that is purely state-based. For every state s, we use a sufficient criterion for deciding whether the given landmark *might have been achieved* on the path from the initial state to s. If yes, s is considered as a goal state in the modified task and hence will be assigned a heuristic value of 0 by the associated abstraction heuristic.

Without path information, how can we decide whether a given landmark could have been reached prior to state s? The key to this question is the notion of a *possibly-before* set for facts of delete relaxations, which has been previously considered by Porteous and Cresswell (2002). We say that a fact f' is *possibly before* fact f if f' can be achieved in the delete relaxation of the planning task without achieving f. We write $pb(f)$ for the set of facts that are possibly before f; this set can be efficiently computed using a fixpoint computation shown in Alg. 1 (function POSSIBLYBEFORE). From the monotonicity properties of delete relaxations, it follows that if l is a delete-relaxation landmark and all facts of the current state s are contained in $pb(l)$, then l still has to be achieved from s.

Based on this insight, the modified task for landmark l can be constructed as follows. First, we compute $pb(l)$. The modified task only contains the facts in $pb(l)$ and l itself; all other facts are removed. The landmark l is the only goal. The initial state and operators are identical to the original task, except that we discard operators whose preconditions are not contained in $pb(l)$ (by the definition of possibly-before sets, these can only become applicable after reaching l) and for all operators that achieve l, we make l their only effect. (Adapting such operators is necessary because they might have other effects that fall outside $pb(l)$. Note that such operators are guaranteed to achieve a goal state, and for an abstraction heuristic it does not matter which exact goal state we end up in.) The complete construction is shown in Alg. 1.

We write $\mathcal{S}(l)$ for the set of states of the modified task. These are exactly the states s of the original planning task where $s \subseteq pb(l) \cup \{l\}$. The abstraction function that is associated with the modified task maps every state in $\mathcal{S}(l)$ to itself. In all other states the landmark might potentially have been achieved, so they should be mapped to an arbitrary goal state of the modified task. We remark that this mapping is easy to represent within the framework of Cartesian abstrac-

Algorithm 1 Construct modified task for landmark $\langle v, d \rangle$.

function LANDMARKTASK($\Pi, \langle v, d \rangle$)
  $\langle \mathcal{V}, \mathcal{O}, c, s_0, s_\star \rangle \leftarrow \Pi$
  $\mathcal{V}' \leftarrow \mathcal{V}$
  $F \leftarrow$ POSSIBLYBEFORE(Π, $\langle v, d \rangle$)
  for all $v' \in \mathcal{V}'$ **do**
    $\mathcal{D}'(v') \leftarrow \{d' \in \mathcal{D}(v') \mid \langle v', d' \rangle \in F \cup \{\langle v, d \rangle\}\}$
  $\mathcal{O}' \leftarrow \{o \in \mathcal{O} \mid \mathit{pre}(o) \subseteq F\}$
  for all $o \in \mathcal{O}'$ **do**
    if $\langle v, d \rangle \in \mathit{eff}(o)$ **then**
      $\mathit{eff}(o) \leftarrow \{\langle v, d \rangle\}$
  return $\langle \mathcal{V}', \mathcal{O}', c, s_0, \{\langle v, d \rangle\} \rangle$

function POSSIBLYBEFORE($\Pi, \langle v, d \rangle$)
  $\langle \mathcal{V}, \mathcal{O}, c, s_0, s_\star \rangle \leftarrow \Pi$
  $F \leftarrow s_0$
  while F has not reached a fixpoint **do**
    for all $o \in \mathcal{O}$ **do**
      if $\langle v, d \rangle \notin \mathit{eff}(o) \wedge \mathit{pre}(o) \subseteq F$ **then**
        $F \leftarrow F \cup \mathit{eff}(o)$
  return F

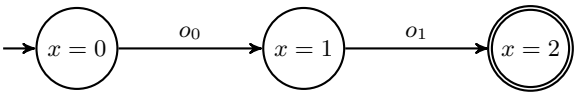

Figure 2: Example task in which operators o_0 and o_1 change the value of the single variable x from its initial value 0 to 1 and from 1 to its desired value 2.

tion (Seipp and Helmert 2013) because $\mathcal{S}(l)$ is a Cartesian set and its complement can be represented as the disjoint union of a small number of Cartesian sets (bounded by the number of state variables of the planning task). Hence the modified task construction can be easily integrated into the Cartesian CEGAR framework.

Abstraction by Landmarks: Improved

In the basic form just presented, the tasks constructed for fact landmarks do not provide as much diversification as we would desire. We illustrate the issue with the example task depicted in Figure 2. The task has three landmarks $x = 0$, $x = 1$ and $x = 2$ that must be achieved in exactly this order in every plan. When we compute the abstraction for $x = 1$, the underlying CEGAR algorithm has to find a plan for getting from $x = 0$ to $x = 1$. Similarly, the abstraction procedure for $x = 2$ has to return a solution that takes us from $x = 0$ to $x = 2$. Since going from $x = 0$ to $x = 2$ includes the subproblem of going from $x = 0$ to $x = 1$, we have to find a plan from $x = 0$ to $x = 1$ twice, which runs counter to our objective of finding abstractions that focus on different aspects on the planning task.

To alleviate this issue, we propose an alternative construction for the planning task for landmark l. The key idea is that we employ a further abstraction that reflects the intuition that at the time we achieve l, certain other landmarks have already been achieved.

In detail, the alternative construction proceeds as follows. We start by performing the basic landmark task construction described in Alg. 1, resulting in a planning task for landmark l which we denote by Π_l.

Furthermore, we use a sound algorithm for computing *landmark orderings* (e. g., Hoffmann, Porteous, and Sebastia 2004; Richter, Helmert, and Westphal 2008) to determine a set L' of landmarks that must necessarily be achieved before l. Note that, unless l is a landmark that is already satisfied in the initial state (a trivial case we can ignore because the Cartesian abstraction heuristic is identical to 0 in this case), L' contains at least one landmark for each variable of the planning task because initial state facts are landmarks that must be achieved before l.

Finally, we perform a *domain abstraction* (Hernádvölgyi and Holte 2000) that combines, for each variable v', all the facts $\langle v', d' \rangle \in L'$ based on the same variable into a single fact.

For example, consider the landmark $l = \langle x, 2 \rangle$ in the above example. We detect that $\langle x, 0 \rangle$ and $\langle x, 1 \rangle$ are landmarks that must be achieved before l. They both refer to the variable x, so we combine the values 0 and 1 into a single value. The effect of this is that in the task for l, we no longer need to find a subplan from $x = 0$ to $x = 1$.

Experiments

We implemented additive Cartesian abstractions in the Fast Downward system and compare them to state-of-the-art abstraction heuristics already present in the planner: h^{iPDB} (Haslum et al. 2007; Sievers, Ortlieb, and Helmert 2012), the two merge-and-shrink heuristics that competed as components of planner portfolios in the IPC 2011 sequential optimization track, $h_1^{\text{m\&s}}$ and $h_2^{\text{m\&s}}$ (Nissim, Hoffmann, and Helmert 2011) and the h^{CEGAR} heuristic (Seipp and Helmert 2013) using a single abstraction. These heuristics can be found in the left part of Table 3.

In the middle part we evaluate three different ways of creating subproblems: abstraction by goals ($h_{s_\star}^{\text{CEGAR}}$), abstraction by landmarks ($h_{\text{LM}}^{\text{CEGAR}}$) and improved abstraction by landmarks ($h_{\text{LM+}}^{\text{CEGAR}}$). The right-most part of Table 3 will be discussed below.

We applied a time limit of 30 minutes and memory limit of 2 GB and let all versions that use CEGAR refine for at most 15 minutes. For the additive CEGAR versions we distributed the refinement time equally among the abstractions.

Table 3 shows the number of solved instances for the compared heuristics on all supported IPC domains. If we look at the results for h^{CEGAR} and $h_{s_\star}^{\text{CEGAR}}$ we see that decomposing the task by goals and finding multiple abstractions separately instead of using only a single abstraction raises the number of solved problems from 562 to 627. This big improvement is due to the fact that all domains except Mprime and Sokoban profit from using $h_{s_\star}^{\text{CEGAR}}$. While h^{CEGAR} has a lower total coverage than h^{iPDB} and $h_2^{\text{m\&s}}$, $h_{s_\star}^{\text{CEGAR}}$ performs better than all compared abstraction heuristics from the literature.

Coverage	h^{iPDB}	$h^{\text{m\&s}}_1$	$h^{\text{m\&s}}_2$	h^{CEGAR}	$h^{\text{CEGAR}}_{s_\star}$	$h^{\text{CEGAR}}_{\text{LM}}$	$h^{\text{CEGAR}}_{\text{LM+}}$	$h^{\text{CEGAR}}_{\text{LM+}s_\star}$ **random**	$h^{\text{add}}\uparrow$	$h^{\text{add}}\downarrow$
airport (50)	21	22	15	19	31	31	27	33	**36**	32
barman-11 (20)	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
blocks (35)	**28**	**28**	20	18	18	18	18	18	18	18
depot (22)	**7**	**7**	6	4	5	4	6	**7**	4	6
driverlog (20)	**13**	12	12	10	10	10	10	10	12	11
elevators-08 (30)	20	1	12	16	**21**	20	17	17	18	18
elevators-11 (20)	16	0	10	13	**18**	17	15	15	15	15
floortile-11 (20)	2	2	**7**	2	2	2	2	2	2	2
freecell (80)	20	16	3	15	15	15	29	28	27	**50**
grid (5)	**3**	2	**3**	2	2	2	2	2	2	2
gripper (20)	7	7	**20**	7	7	7	7	7	7	7
logistics-00 (28)	**21**	16	20	14	20	**21**	16	20	20	20
logistics-98 (35)	4	4	5	3	6	6	6	7	6	**8**
miconic (150)	55	50	**74**	55	67	68	68	69	69	69
mprime (35)	23	23	11	**27**	25	25	25	25	25	26
mystery (30)	16	16	7	**17**	**17**	**17**	**17**	**17**	**17**	**17**
nomystery-11 (20)	16	12	**18**	10	16	14	14	14	14	14
openstacks-06 (30)	7	7	7	7	7	7	7	7	**9**	7
openstacks-08 (30)	**19**	9	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**
openstacks-11 (20)	**14**	4	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
parcprinter-08 (30)	12	15	17	11	13	12	17	17	18	**22**
parcprinter-11 (20)	8	11	13	7	9	8	13	13	14	**17**
parking-11 (20)	**5**	**5**	0	0	0	0	0	0	0	0
pathways (30)	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
pegsol-08 (30)	6	2	**29**	27	27	27	27	27	27	27
pegsol-11 (20)	0	0	**19**	17	17	17	17	17	17	17
pipesworld-nt (50)	**17**	15	8	14	15	16	15	15	15	15
pipesworld-t (50)	**17**	16	7	11	12	12	12	12	12	12
psr-small (50)	49	**50**	49	49	49	49	48	49	49	49
rovers (40)	7	6	**8**	6	7	7	7	7	7	7
satellite (36)	6	6	**7**	6	6	6	6	6	6	6
scanalyzer-08 (30)	**13**	6	12	12	12	12	12	**13**	**13**	12
scanalyzer-11 (20)	**10**	3	9	9	9	9	9	**10**	**10**	9
sokoban-08 (30)	**29**	3	23	21	20	21	22	22	23	22
sokoban-11 (20)	**20**	1	19	18	17	18	19	19	19	19
tidybot-11 (20)	**14**	13	0	13	**14**	**14**	**14**	**14**	**14**	**14**
tpp (30)	6	6	7	6	**10**	7	6	7	7	7
transport-08 (30)	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
transport-11 (20)	6	6	**7**	6	6	6	6	6	6	6
trucks (30)	8	6	8	6	9	10	9	11	**12**	**12**
visitall-11 (20)	**16**	**16**	9	9	9	9	9	9	**16**	9
wood-08 (30)	7	**14**	9	9	10	11	11	11	11	11
wood-11 (20)	2	**9**	4	5	5	6	6	6	6	6
zenotravel (20)	11	9	**12**	9	**12**	**12**	**12**	**12**	**12**	**12**
Sum (1396)	600	475	578	562	627	625	635	653	667	**685**

Table 3: Number of solved tasks by domain for different heuristics. Best values are highlighted in bold.

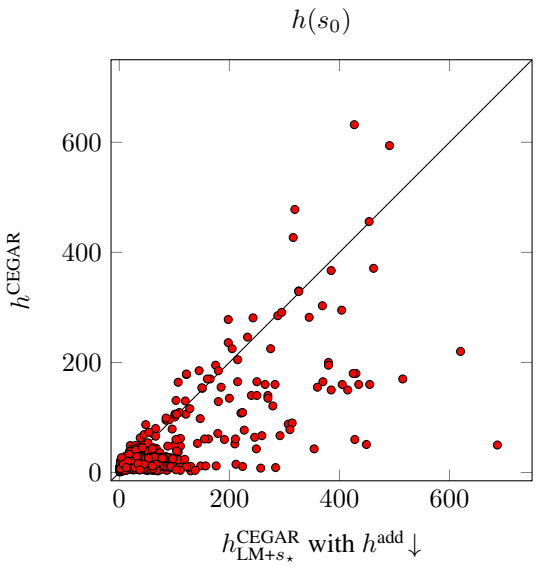

Figure 3: Comparison of the heuristic estimates for the initial state made by h^{CEGAR} and $h^{\text{CEGAR}}_{\text{LM}+s_\star}$ with $h^{\text{add}}\downarrow$ on the tasks from Table 3. We omit the results for the ParcPrinter domain since it has much higher costs. Points below the diagonal represent tasks for which $h^{\text{CEGAR}}_{\text{LM}+s_\star}$ with $h^{\text{add}}\downarrow$ makes a better estimate than h^{CEGAR}.

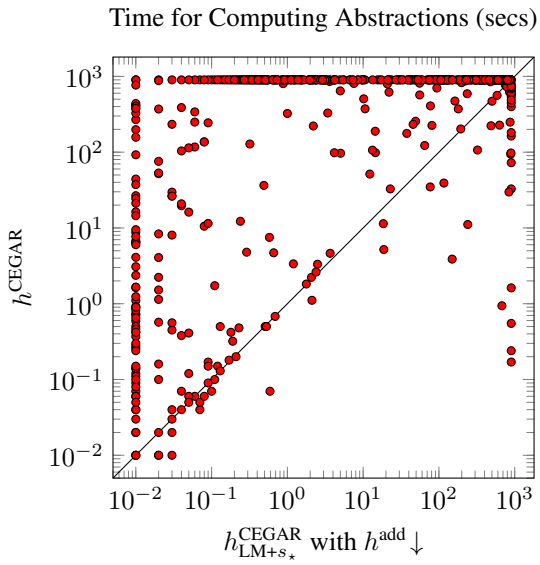

Figure 4: Comparison of the time taken by h^{CEGAR} and $h^{\text{CEGAR}}_{\text{LM}+s_\star}$ with $h^{\text{add}}\downarrow$ to compute all abstractions on the tasks from Table 3. Points above the diagonal represent tasks for which $h^{\text{CEGAR}}_{\text{LM}+s_\star}$ with $h^{\text{add}}\downarrow$ needs less time for the computation than h^{CEGAR}.

In total $h^{\text{CEGAR}}_{s_\star}$ and $h^{\text{CEGAR}}_{\text{LM}}$ solve roughly the same number of problems (627 and 625) and also for the individual domains coverage does not change much between the two heuristics. Only when we employ the improved $h^{\text{CEGAR}}_{\text{LM+}}$ heuristic that uses domain abstraction to avoid duplicate work during the refinement process, the number of solved problems increases to 635.

Abstraction by Landmarks and Goals

We can observe that $h^{\text{CEGAR}}_{s_\star}$ and $h^{\text{CEGAR}}_{\text{LM+}}$ outperform each other on many domains: for maximum coverage $h^{\text{CEGAR}}_{s_\star}$ is preferable on 7 domains, whereas $h^{\text{CEGAR}}_{\text{LM+}}$ should be preferred on 9 domains. This suggests trying to combine the two approaches.

We do so by first computing abstractions for all subproblems returned by the *abstraction by landmarks* method. If afterwards the refinement time has not been consumed, we also calculate abstractions for the subproblems returned by the *abstraction by goals* decomposition strategy for the remaining time. The results for this approach ($h^{\text{CEGAR}}_{\text{LM}+s_\star}$-random) are shown in the third from last column in Table 3. Not only does this approach solve as many problems as the better performing ingredient technique in many individual domains, but it often even outperforms both original diversification methods, raising the total number of solved tasks to 653.

Subproblem Orderings

Since the cost saturation algorithm is influenced by the order in which the subproblems are considered, we evaluate the impact of three different orderings. All previously discussed results were obtained with the *random* ordering that randomly shuffles the subproblems before passing them to the cost saturation algorithm. We hypothesize that it could be beneficial to order the subproblems either with ascending or descending difficulty. Therefore, we test two orderings $h^{\text{add}}\uparrow$ and $h^{\text{add}}\downarrow$ that order the subproblems by the h^{add} value (Bonet and Geffner 2001) of the corresponding goal fact or landmark. This allows us to work on facts closer to the initial state or closer to the goal first.

The results for the different orderings with $h^{\text{CEGAR}}_{\text{LM}+s_\star}$ can be seen in the three right-most columns in Table 3. The random ordering solves more tasks than the two ordering methods based on h^{add} values only in the Depot domain. Everywhere else at least one of the two principled sortings solves at least as many tasks as the random one. However, none of them outperforms the other on all domains. $h^{\text{add}}\uparrow$ solves more tasks than $h^{\text{add}}\downarrow$ in 7 domains but $h^{\text{add}}\downarrow$ also has a higher coverage than $h^{\text{add}}\uparrow$ in 6 domains. Both principled orderings raise the total coverage over the random one: $h^{\text{add}}\uparrow$ solves 667 problems whereas $h^{\text{add}}\downarrow$ finds the solution for 685 tasks.

Our new best method $h^{\text{CEGAR}}_{\text{LM}+s_\star}$ with $h^{\text{add}}\downarrow$ solves 123 more tasks than our previous method h^{CEGAR} (685 vs. 562). This coverage improvement of 21.9% is substantial because in most domains solving an additional task optimally becomes exponentially more difficult as the tasks get larger.[1]

[1]For comparison, if we look at the non-portfolio planners in IPC 2011 (sequential optimization track), the best one only solved 1.8% more problems than the fourth-best one. If we also include portfolio systems, the winner solved "only" 11.4% more problems than the 6th-placed system.

The big increase in coverage can be explained by the fact that $h^{\text{CEGAR}}_{\text{LM}+s_\star}$ with $h^{\text{add}}\downarrow$ estimates the solution cost much better than h^{CEGAR} as shown in Figure 3. One might expect that the increased informedness would come with a time penalty, but in Figure 4 we can see that in fact $h^{\text{CEGAR}}_{\text{LM}+s_\star}$ with $h^{\text{add}}\downarrow$ takes *less* time to compute the abstractions than h^{CEGAR}. Since all individual CEGAR invocations only stop if they run out of time or find a concrete solution, Figure 4 tells us that for most tasks h^{CEGAR} does not find a solution, but instead uses the full 15 minutes for the refinement whereas $h^{\text{CEGAR}}_{\text{LM}+s_\star}$ with $h^{\text{add}}\downarrow$ almost always needs less time.

Conclusion

We presented an algorithm that computes a cost partitioning for explicitly represented abstraction heuristics and showed that it performs best when invoked for complementary abstractions. To this end, we introduced several methods for generating diverse abstractions. Experiments show that the derived heuristics often outperform not only the single Cartesian abstractions, but also many other state-of-the-art abstraction heuristics.

Future research could try to use our cost saturation algorithm to build additive versions of other abstraction heuristics such as pattern databases or merge-and-shrink abstractions.

Acknowledgments

The Swiss National Science Foundation (SNSF) supported this work as part of the project "Abstraction Heuristics for Planning and Combinatorial Search" (AHPACS).

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Computational Intelligence* 11(4):625–655.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1):5–33.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-guided abstraction refinement. In Emerson, E. A., and Sistla, A. P., eds., *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, 154–169.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

Hernádvölgyi, I. T., and Holte, R. C. 2000. Experiments with automatically created memory-based heuristics. In Choueiry, B. Y., and Walsh, T., eds., *Proceedings of the 4th International Symposium on Abstraction, Reformulation and Approximation (SARA 2000)*, volume 1864 of *Lecture Notes in Artificial Intelligence*, 281–290. Springer-Verlag.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Holte, R.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence* 170(16–17):1123–1136.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1728–1733.

Katz, M., and Domshlak, C. 2008. Optimal additive composition of abstraction-based admissible heuristics. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 174–181. AAAI Press.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, 335–340. IOS Press.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1983–1990.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2357–2364.

Porteous, J., and Cresswell, S. 2002. Extending landmarks analysis to reason about resources and repetition. In *Proceedings of the 21st Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG '02)*, 45–54.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 975–982. AAAI Press.

Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 347–351. AAAI Press.

Sievers, S.; Ortlieb, M.; and Helmert, M. 2012. Efficient implementation of pattern database heuristics for classical planning. In Borrajo, D.; Felner, A.; Korf, R.; Likhachev, M.; Linares López, C.; Ruml, W.; and Sturtevant, N., eds., *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SoCS 2012)*, 105–111. AAAI Press.

Yang, F.; Culberson, J.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research* 32:631–662.